Industrial use of formal methods

Lars-Henrik Eriksson
Dept. of Information Technology
(Computing Science)
Uppsala University

lhe@it.uu.se
http://user.it.uu.se/~lhe

2004-09-27

Using ”formal methods” means

…using *mathematically exact* techniques to

• improve quality
• increase productivity

when developing systems performing *computations* in a broad sense.

Such as

• computer software
• electronic control systems
• electromechanical control systems

Traditional software construction

• Describing the task to be performed by the program using natural language (possibly supplemented with tables etc.)
• System design and coding
• Testing and debugging

leads to

• large costs for testing and debugging
• delivering software which still is not bug-free

Different software development methodologies have improved the situation, but not led to decisive improvements.

Mathematical models

Other engineering disciplines would never accept a methodology that relies so heavily on testing.

Instead, designs are analysed using *mathematical models* from e.g.

• material science (e.g. bridge construction)
• control theory (e.g. process control)
• electricity theory (e.g. electronics)

… giving a much lower probability of incorrect design.

Formal methods is a methodology with a similar role in the development of software and related systems.

Formal methods

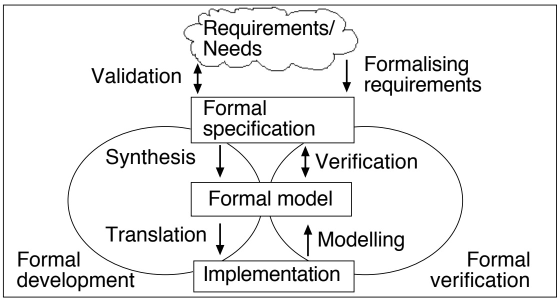
…are based on *formal*, that is

mathematical/logical

• requirements specifications
• design models
• *proofs* of relations between design models and specifications.

Discrete mathematics and *mathematical logic* is used.

Concepts of formal methods



Formal specifications

The formal specification should give an exact description of the requirements of the system. Requirements can concern e.g.:

• Functionality
• Time aspects
• Fault tolerance

Incomplete requirements can still be of good use.

Validating the specification

How do you know that the formal specification correctly represents the necessary requirements?

A good understanding of the formal language used is needed in order to avoid mistakes, but the specification must also be *validated*.

A formal specification can be validated with the aid of a computer using software tools to do:

• Simulation (the computer simulates the specified system)
• Testing (also by *proving* properties of the specification)

Using formal specifications

A formal specification is a prerequisite for using formal methods, e.g. for the verification of programs and systems.

The specifications are of value even if formal methods are not otherwise used. Among other things, they can be used to

• Improve understanding of requirements.
• Improve communication of requirements.
• Automatically construct test data.

A large part of software errors are caused by poor and/or incorrect requirements specifications.

Formal verification

Formally verifying the system means proving that the model of the implementation fulfils the requirements as expressed by the formal specification.

Since proofs are exact, a successful proof ensures that the requirements are satisfied (provided that the proof has been carried out correctly).

Proofs can be carried out

• By hand
• With computer support
• Automatically

Formal development

…means that the implementation is constructed formally from the specification.

Typically the end result is in a formal notation that can be automatically translated into ”ordinary” code (C, Ada etc.)

If every step in the construction process is correct, the result is guaranteed to fulfil the requirements.

Formal development is a more difficult problem than verification, since there is not a single correct solution.

Fully automated development is not possible in practise (but you can get close… se Siemens experience further on)

Z / B / VDM-SL

Z, B and VDM-SL are ”model-oriented” specification languages.

Z and B are based on Zermelo-Fraenkel set theory.

B was developed from Z by J.R. Abrial with the intention of making a more practically useful system for formal development.

VDM-SL is based on the logic of partial functions. Many concepts are similar, but VDM-SL has a three valued logic.

VDM-SL is a result of IBM research on formal semantics for the programming language PL/1.

A controlled experiment

Few studies done on how FM affect software development projects? Controlled experiments difficult: high cost and risk.

British Aerospace Systems and Equipment Ltd (BASE) study in the first half of the 1990s:

Develop a simple (but realistic!) security-critical program *twice* – with and without formal specifications (VDM-SL) being used.

Traditional development process using the "V" model.
Input – a natural language customer requirements document.
Output – software written in C + various development metrics.

Same resources to both efforts. Similar staff qualifications.

General results

Both teams delivered within schedule and budget.

Engineers had little trouble learning and using VDM-SL.
Tool support essential!

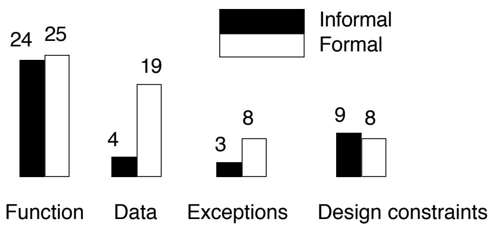
System analysis took longer in the formal project.
Design and implementation took less time.

Informal development produced slow, poor-quality software because of a mistake detected late in the project.
Strict quality standards would force development to start over!

No certain conclusions can be drawn, but this all concurs with "accepted wisdom" in the FM community.

Queries about customer req's

Queries were logged and analysed. Formal team made 50% as many queries as informal team.



How (are) formal spec. used in industry?

- Not much, but steadily increasing
- Greatest acceptance with safety-critical applications and digital circuit design (remember the Pentium I floating-point division error about 10 years ago).
- Several process and quality standards demand or recommend the use of formal methods (e.g. UK DEF STAN 00-55, Cenelec EN5012x series)

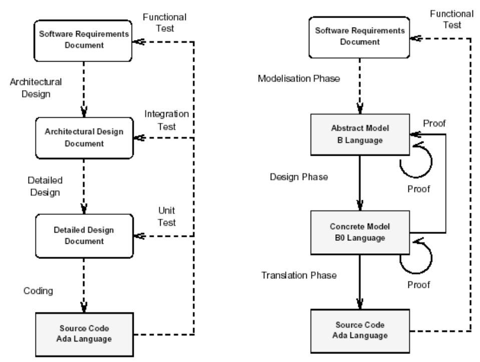
Siemens Transportation Systems
(formely MATRA transport)
use of B

Develops ATP/ATO (Automatic Train Protection/Operation) systems.

Have used the B-method for more than 10 years.

Complete formal development process from specifications to executable code.

Safety-critical part of Météor (software for Paris metro line 14) comprises 86 000 lines of Ada code, all developed formally.



Siemens Transportation Systems some statistics

Project:	Météor (1996) (Paris metro 14)	CBTC (2004) (New York Canarsie Line)
Staff:	10	4
Proof obligiations	23 000	41 000
Automatically proved	75%	86%
Effort, man-years	17	4
Lines of generated code	86 000	> 100 000

Siemens Transportation Systems experiences

≈90% (today) of all proof obligations proved automatically.

Several errors detected during development by failed proof.

Resulting code is virtually error–free.

No testing done until the system test.

Cost of developing safety-critical software approaching that of non safety-critical software.

(Important obstacle to developing non safety-critical software with B is lack of qualified developers.)

Volvo Car Corporation

Current family of cars beginning with the S80 model. Most functions computer-controlled. A large number of computers connected by local area network.

Completely computerised engine controls – no mechanical linkage between accelerator pedal and engine.

Very strict requirements on engine control software due to environmental (legal) and safety constraints.

Software written by subcontractors. Do they really produce what they are supposed to?

Formal specifications (temporal propositional logic) used contractually for the software with very good results.

Bombardier Transportation Rail Control Solutions

Develops railway signalling systems. Main product: Ebilock 850/950 computer interlocking (”Ställverk 85/95” in Sweden).

Different software versions for different railways. Specification traditionally by ”use cases”. >10 000 use cases for each version of the software.

Formal specification notation (predicate logic) used instead.

Formal verification integrated into the development process.

Testing and debugging effort reduced by 90%.

Banverket

(Swedish National Rail Administration)

Contract with Vossloh Signal-Technik (Malmö) to develop a new generation of small railway interlocking systems (”Alister”).

Development according to the new CENELEC standards EN 50126, 50128 and 50129.

Independent validator checks that the developed system complies with the specification.

Validator found that the informal specification was insufficient. Central parts of the specification rewritten formally using an extended state-machine notation.

Banverket (more)

(Swedish National Rail Administration)

Formal verification of Alister software initially revealed many deviations from the specifications.

The software could be corrected before being put into operational test.

Operational testing revealed problems with the non-verified (non-safety critical) software.

Formal Methods do not do everything

- The whole of the development process is not covered by formal methods.

- Not all kinds of questions can be handled with formal methods (in practise)

Testing is still required – but to a lesser extent.